

FIG. 1

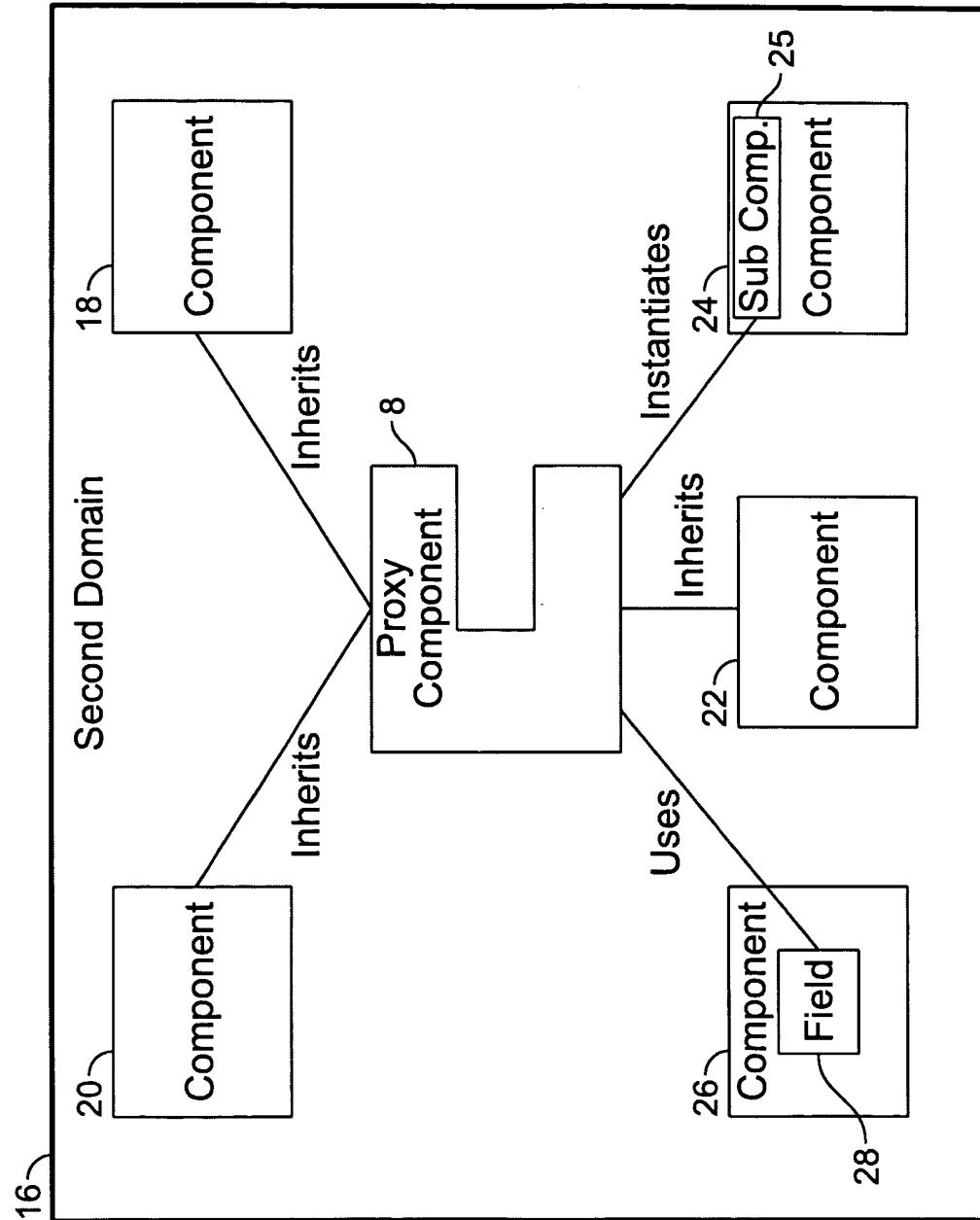


FIG. 2

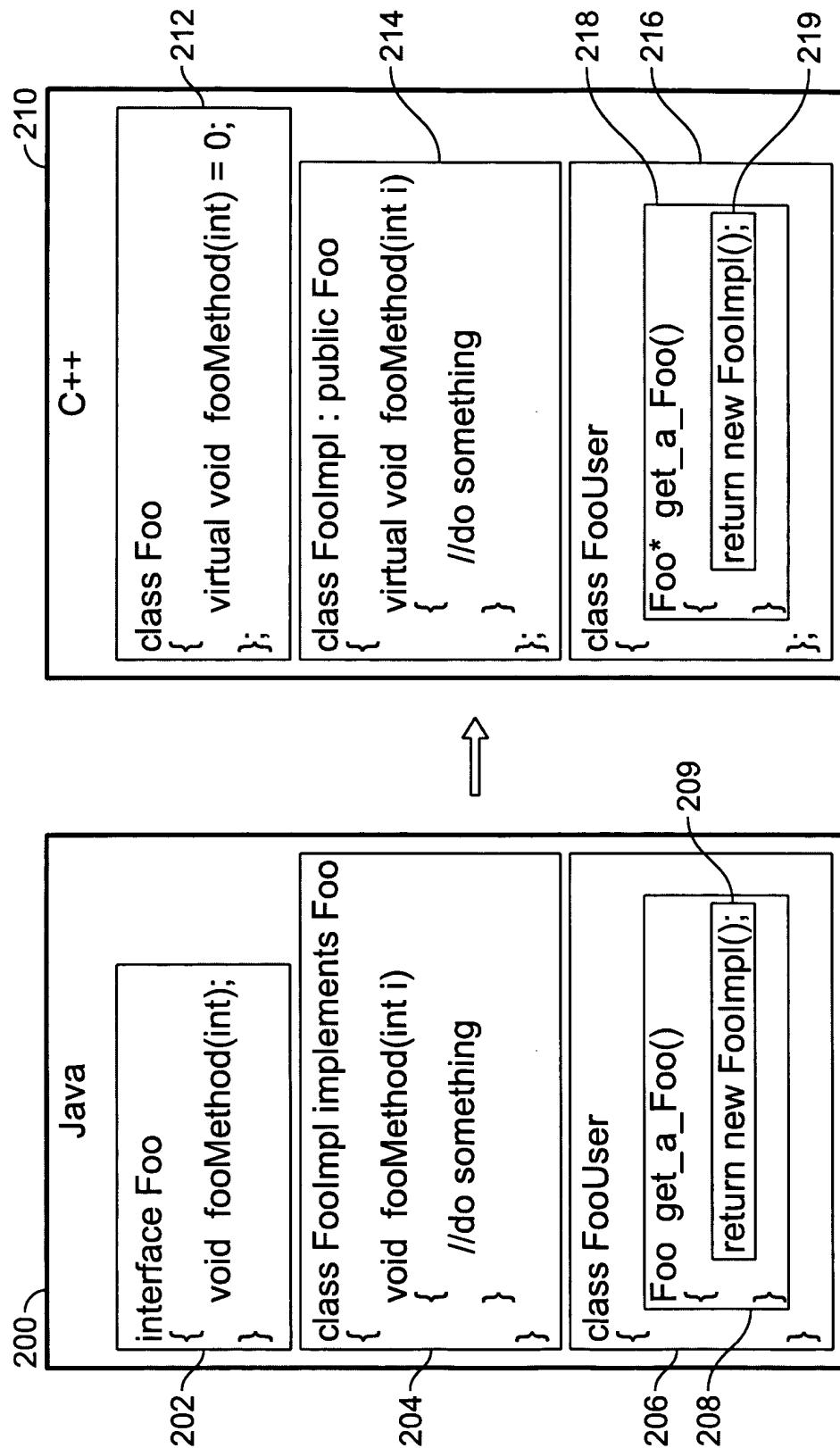


FIG. 3
FIG. 4



220

C++

```
class Foo : protected virtual jcpp_ref  
{  
    Foo( jobject_obj ) : jcpp_ref( _obj )  
    {  
        //some JNI code  
    }  
    virtual void fooMethod(int)  
    {  
        //call the JNI method using virtual  
        //invocation  
    }  
};
```

230
223
224
222

226

```
class FooUser  
,  
{  
    Foo get_a_Foo()  
    {  
        jobject aFoo = ...; //call Java method using virtual JNI  
        invocation  
        return Foo(aFoo);  
    }  
};
```

228
229

FIG. 5



30 proxy_name (jobject_obj, int_type);

36 proxy_name (const proxy_ref* ref, const char* fieldName);

42 proxy_name (const proxy_class* clazz, const char* fieldName);

48 proxy_name (const proxy_array* array, jsize index);

38 40 44 46 50 52

FIG. 6



108

```

class Foo : public java::lang::Object
{
public:
    typedef jcpp_object_array<Foo> array1D; 110
    Foo( const Tnull B); 112 114
    Foo( object, int );
    Foo( const jcpp_ref*, const char* );
    Foo( const jcpp_class*, const char* );
    Foo( const jcpp_array*, jsize );
    Foo( const Foo & ); 116
    ~Foo(); 118 120
    Foo& operator = ( const Foo & );

    bool operator == ( const Foo & ) const; 122
    bool operator != ( const Foo & ) const; 124
    static const jcpp_class* get_static_class(); 126
    const jcpp_class* get_class() const; 128
    static Foo dyna_cast( const jcpp_ref & _src ); 130
};

```

FIG. 7



52

```
class jcpp_int : public jcpp_base {  
public:  
    typedef jcpp_int_array 57 array1D; 58  
    typedef (Tobject_array<array1D>) array2D;  
  
    jcpp_int( const jcpp_ref * _ref, const char * _fieldName );  
    jcpp_int( const jcpp_class * _ref, const char * _fieldName );  
    jcpp_int( const jcpp_int_array * _array, jsize _index ); 61  
    jcpp_int( const jcpp_int & _rns ); 62 59  
    jcpp_int( ); 66  
    operator new ( size_t _size );  
    operator delete( void * _ptr ); 62  
    operator jint ( ) const; 64  
    jcpp_int & operator = ( jint );  
    jcpp_int & operator += ( jint );  
    jcpp_int & operator -= ( jint );  
    jcpp_int & operator *= ( jint );  
    jcpp_int & operator /= ( jint );  
    jcpp_int & operator %= ( jint ); 66  
    jcpp_int & operator ++ ( );  
    jcpp_int & operator -- ( );  
    jint operator ++ ( int );  
    jint operator -- ( int );  
  
    const jcpp_class * get_class() cont; 68  
};
```

FIG. 8

Java		C++	
Declarations		Declarations	
230		250	
232	interface Foo	252	class Foo
233	{ public static final (int X=25); }	253	{ static const jint X;
234			};
235	class Bar	254	class Bar
237	{ static int X=25;	255	{ static jint X;
	char ch = 'a';	257	char ch;
	}		};
		Usage	
236	int i = Foo.X;	256	int i = Foo::X;
238	int j = Bar.X;	258	int j = Bar::X;
240	Bar.X = 30;	260	Bar::X = 30;
242	char k = new Bar().ch;	262	jchar k = Bar() .ch;
244	Bar bar = new Bar();	264	Bar bar;
246	bar.ch = 'd';	266	Bar.ch = 'd';

FIG. 9

FIG. 10



270

C++

Declarations

272 class Foo
273 {
273 static const jcpp_int X;
273 };

274 class Bar
275 {
277 static jcpp_int X;
277 jcpp_char ch;
277 };

Usage

256 int i = Foo::X;
258 int j = Bar::X;
260 Bar::X += 30;
262 jchar k = Bar().ch;
264 Bar bar;
266 Bar.ch = 'd';

FIG. 11

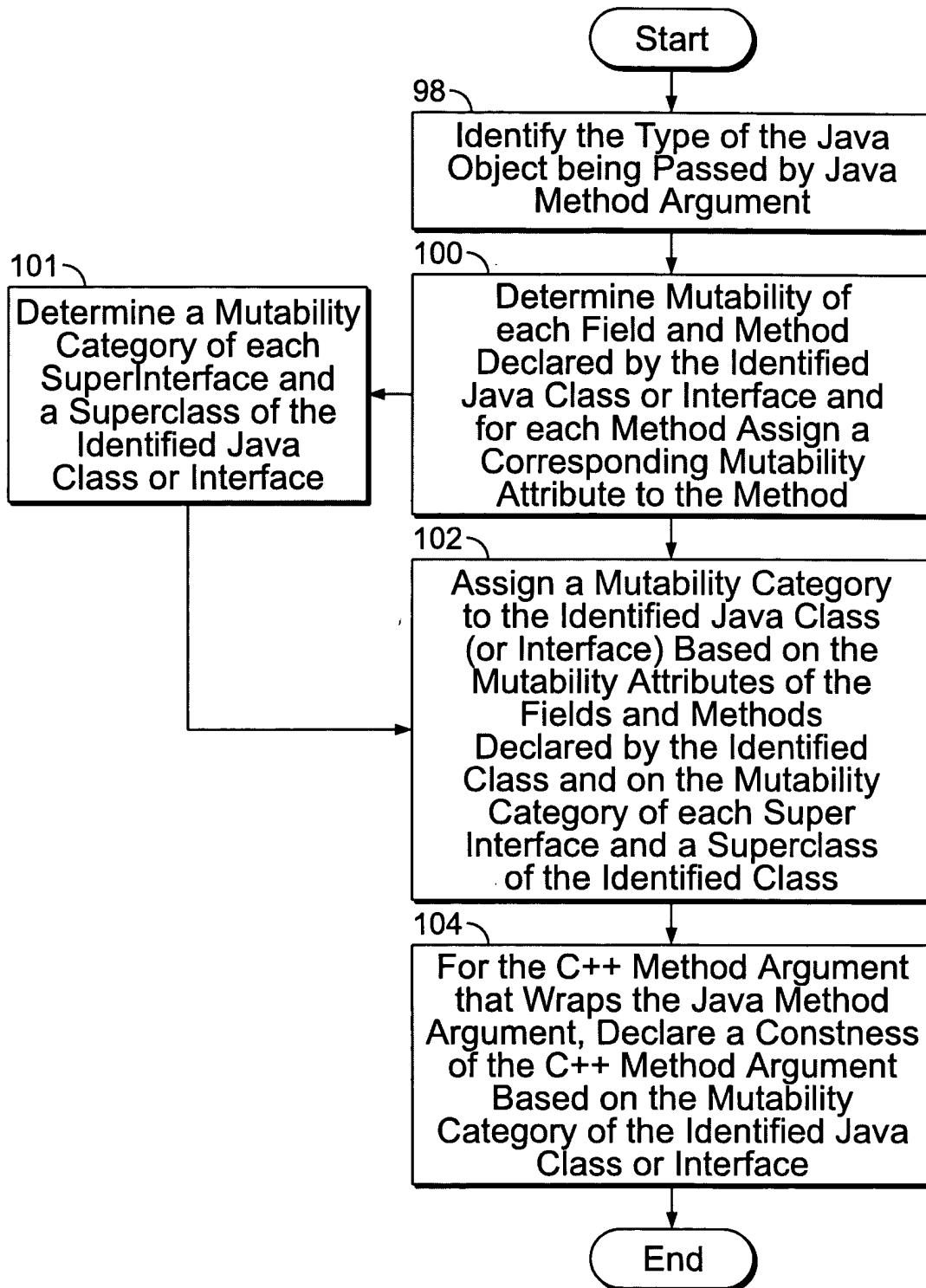


FIG. 12

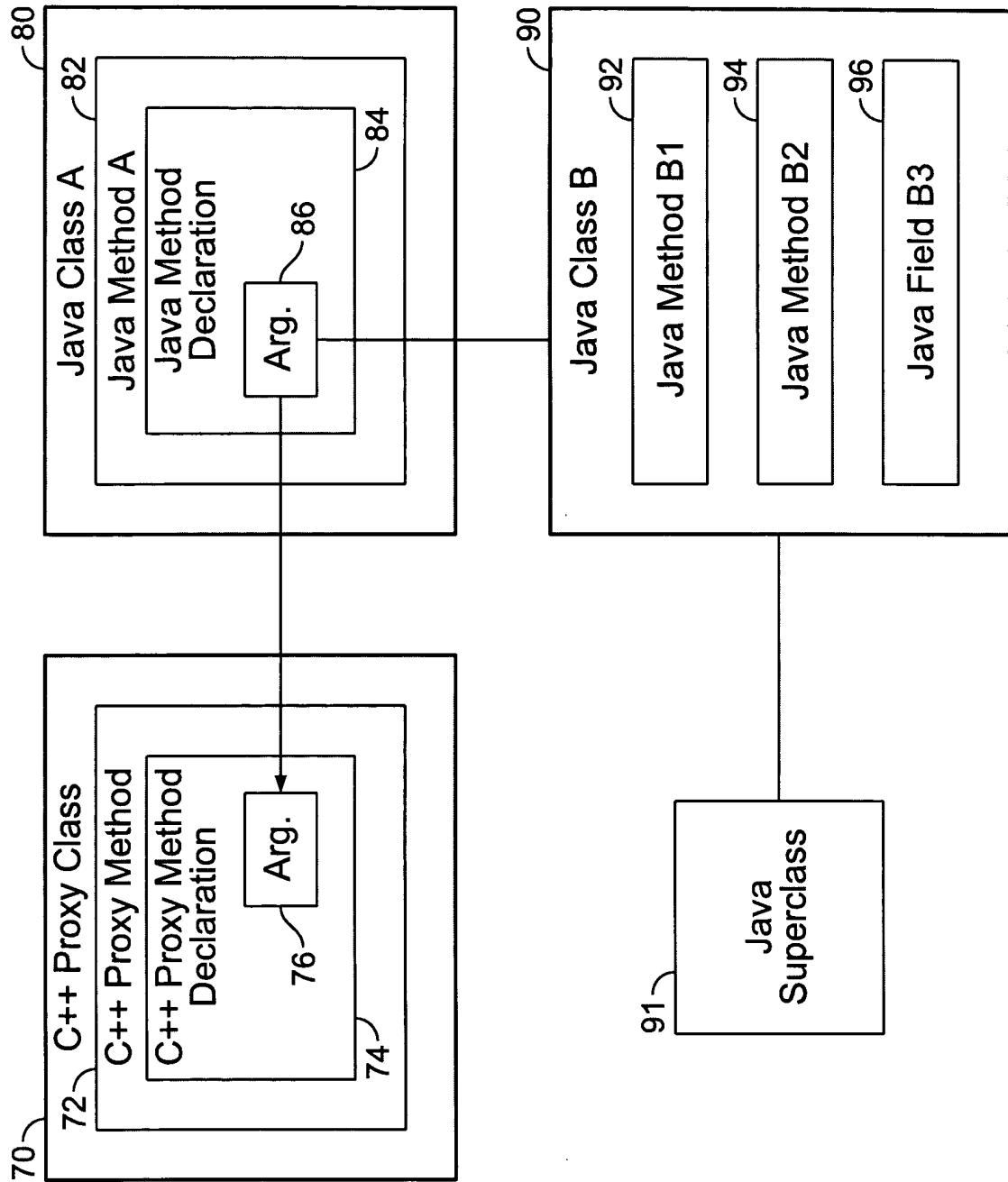


FIG. 13

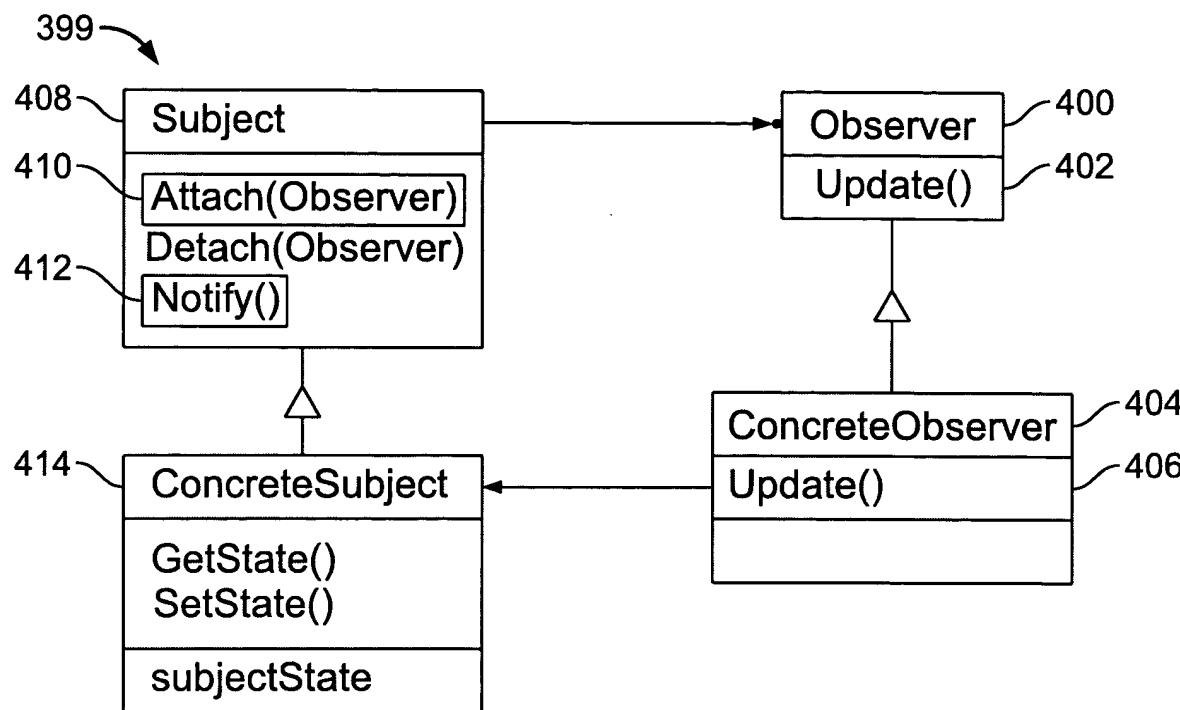
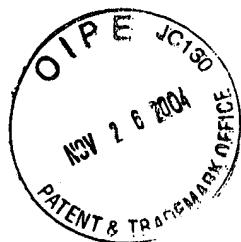


FIG. 14

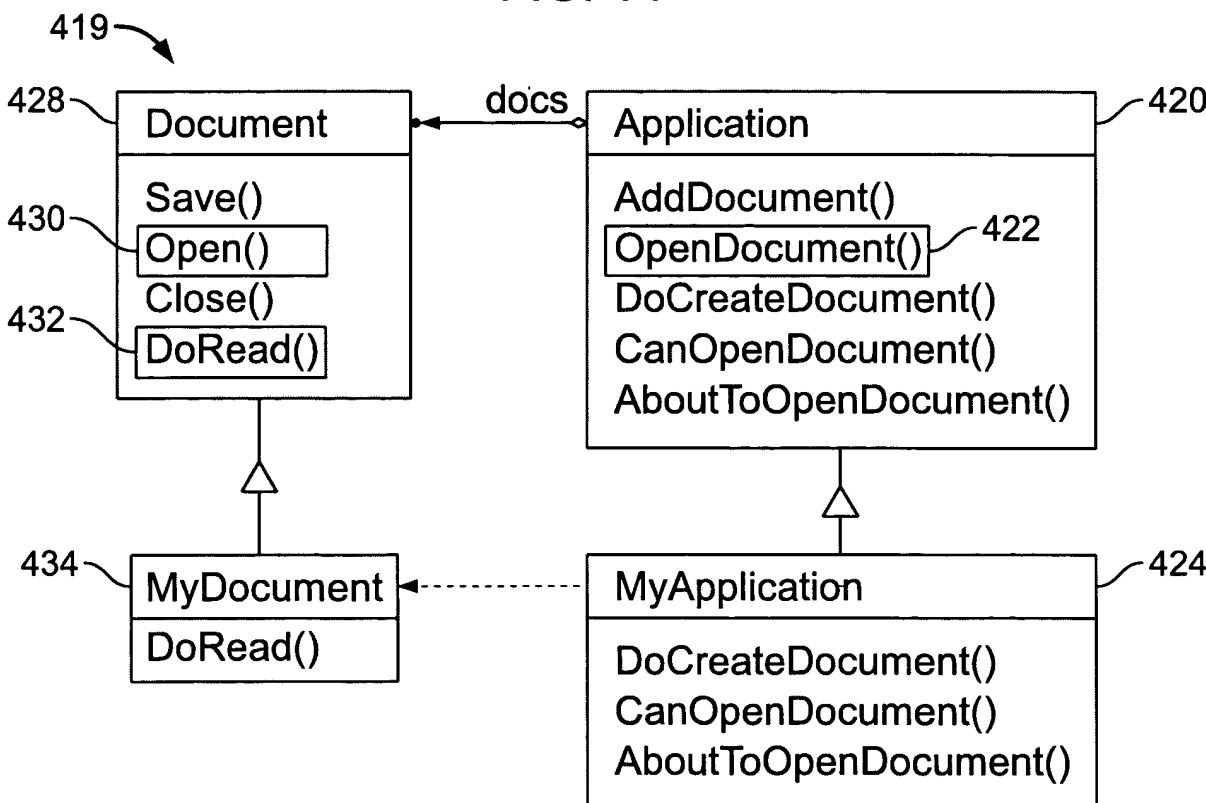


FIG. 15

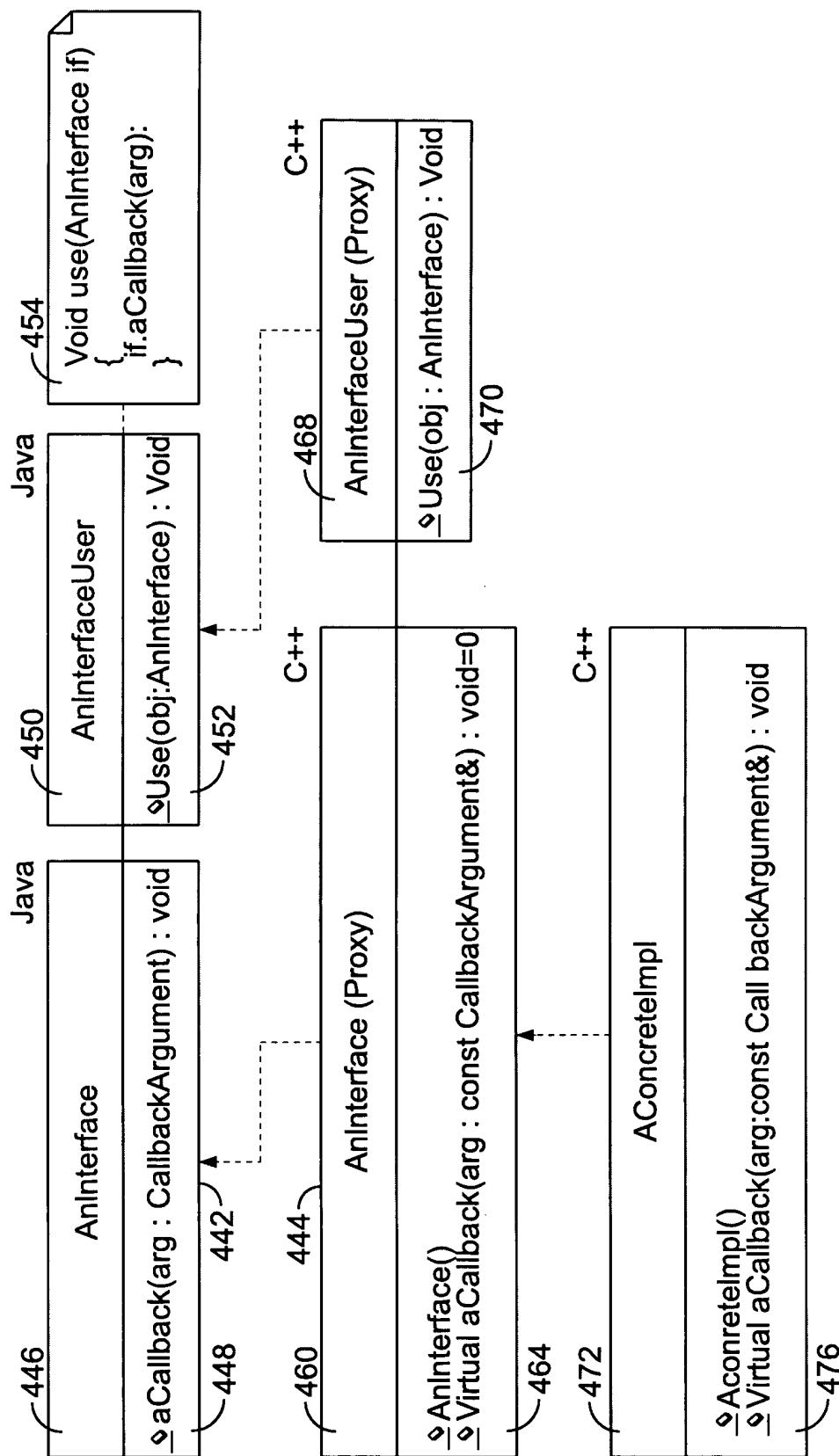


FIG. 16

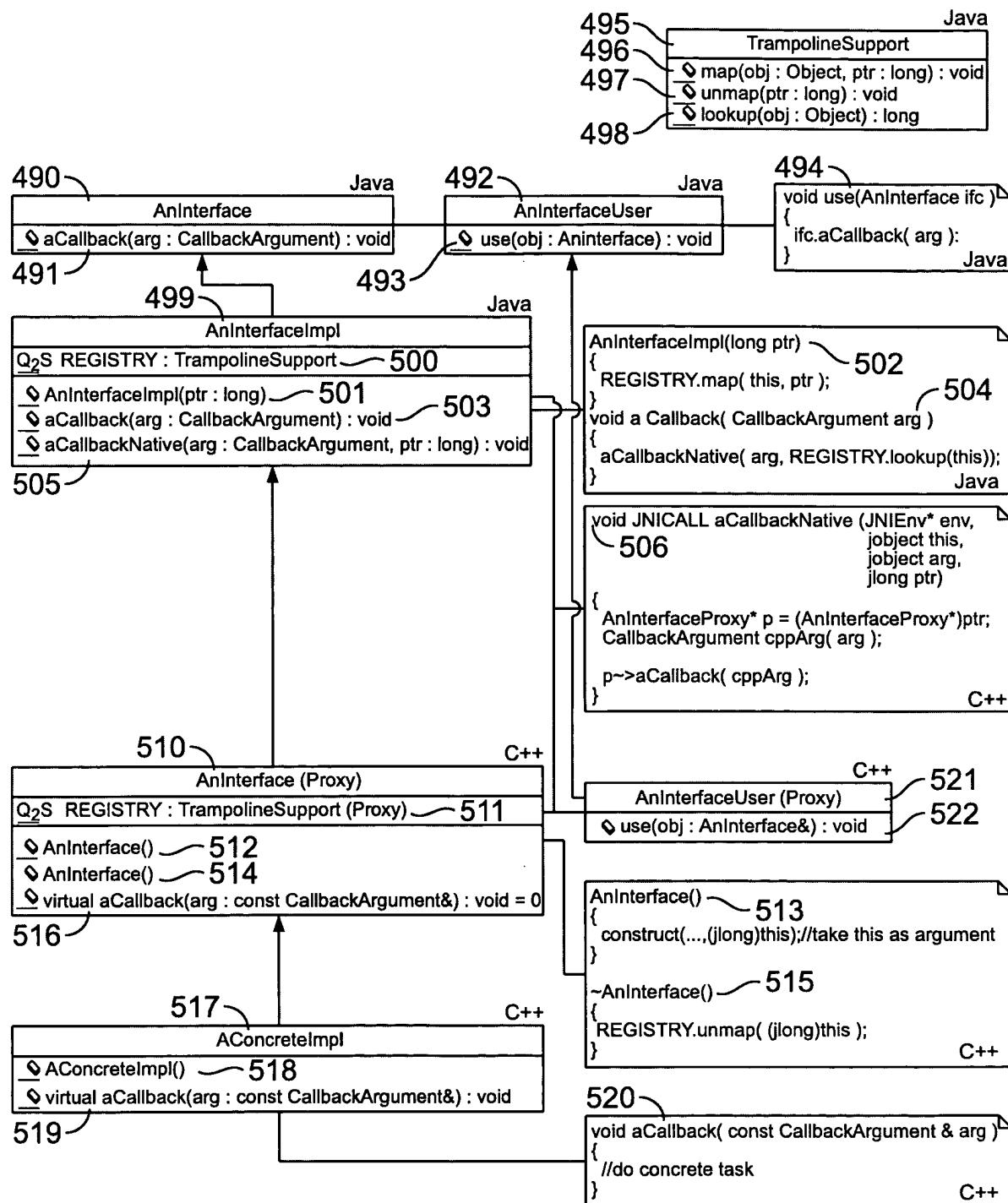


FIG. 17

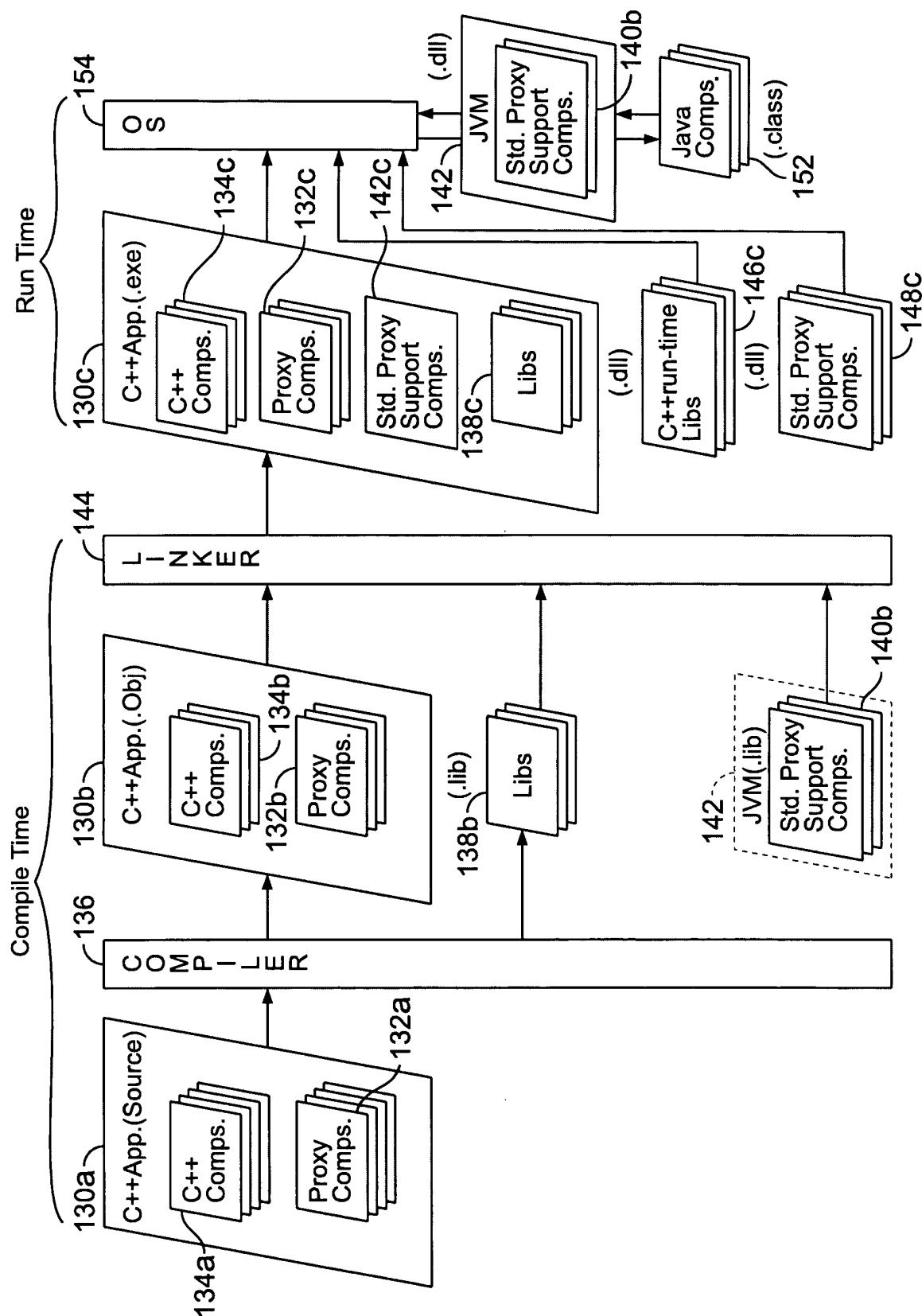


FIG. 18

}

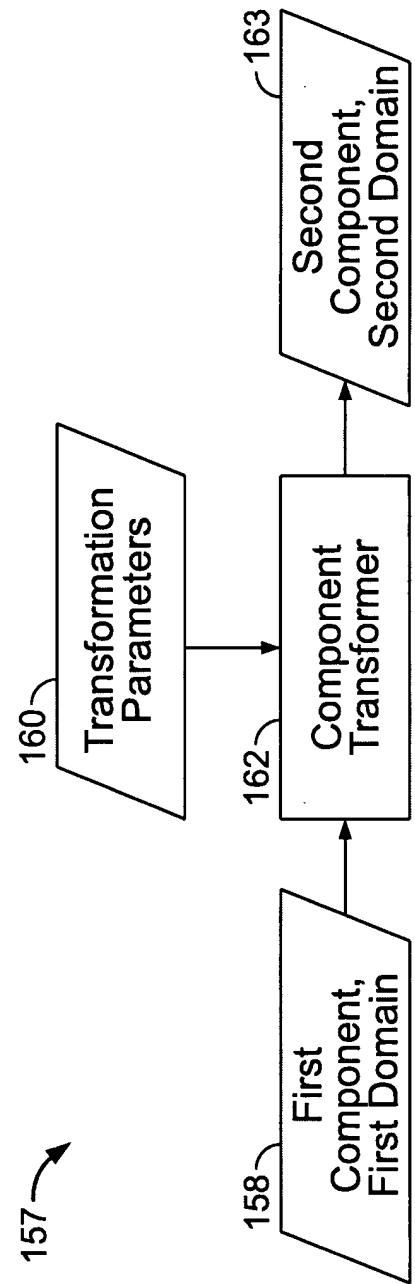


FIG. 19

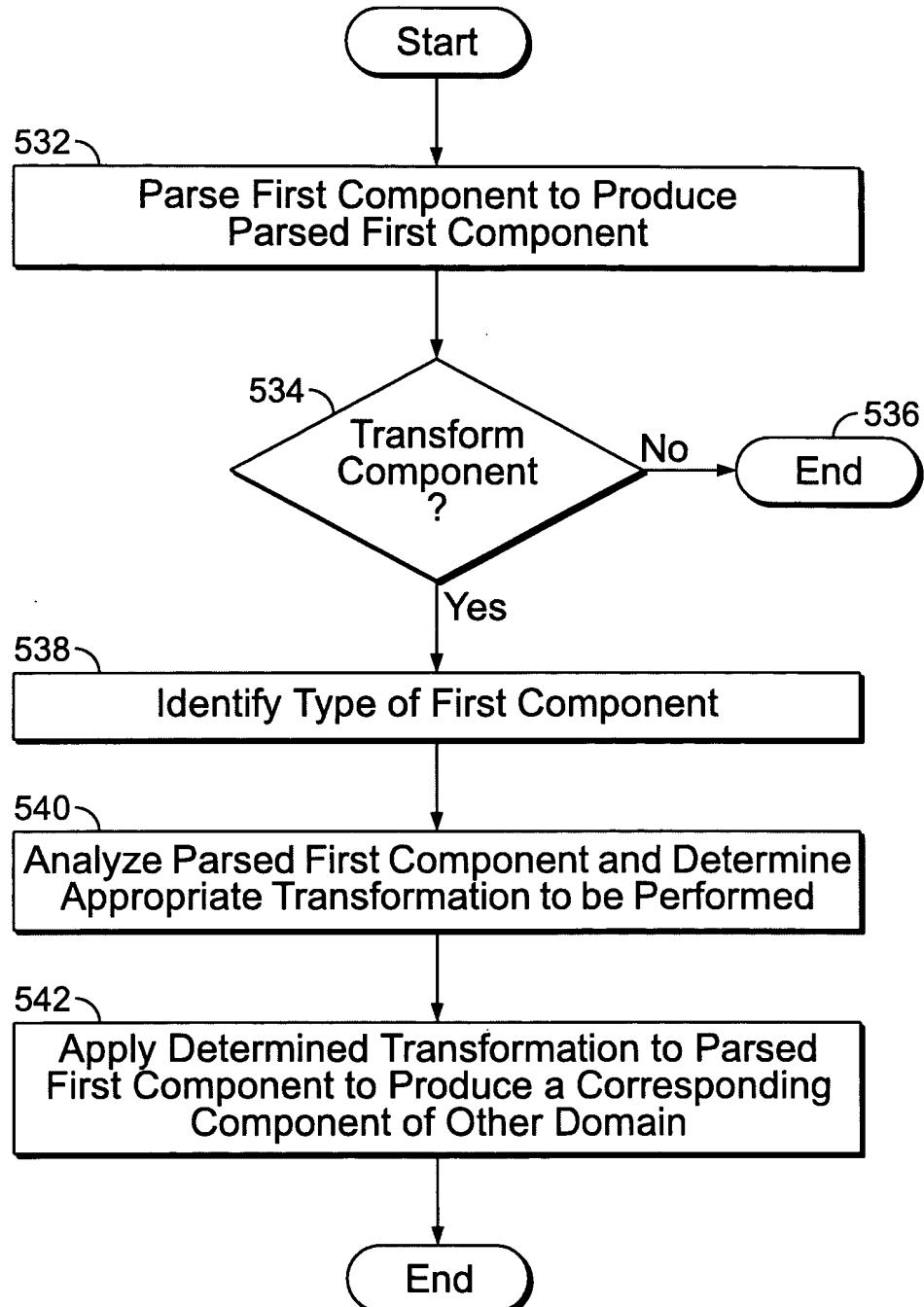
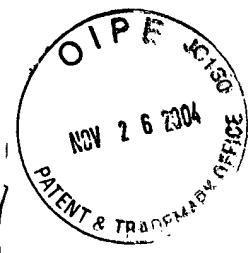


FIG. 20

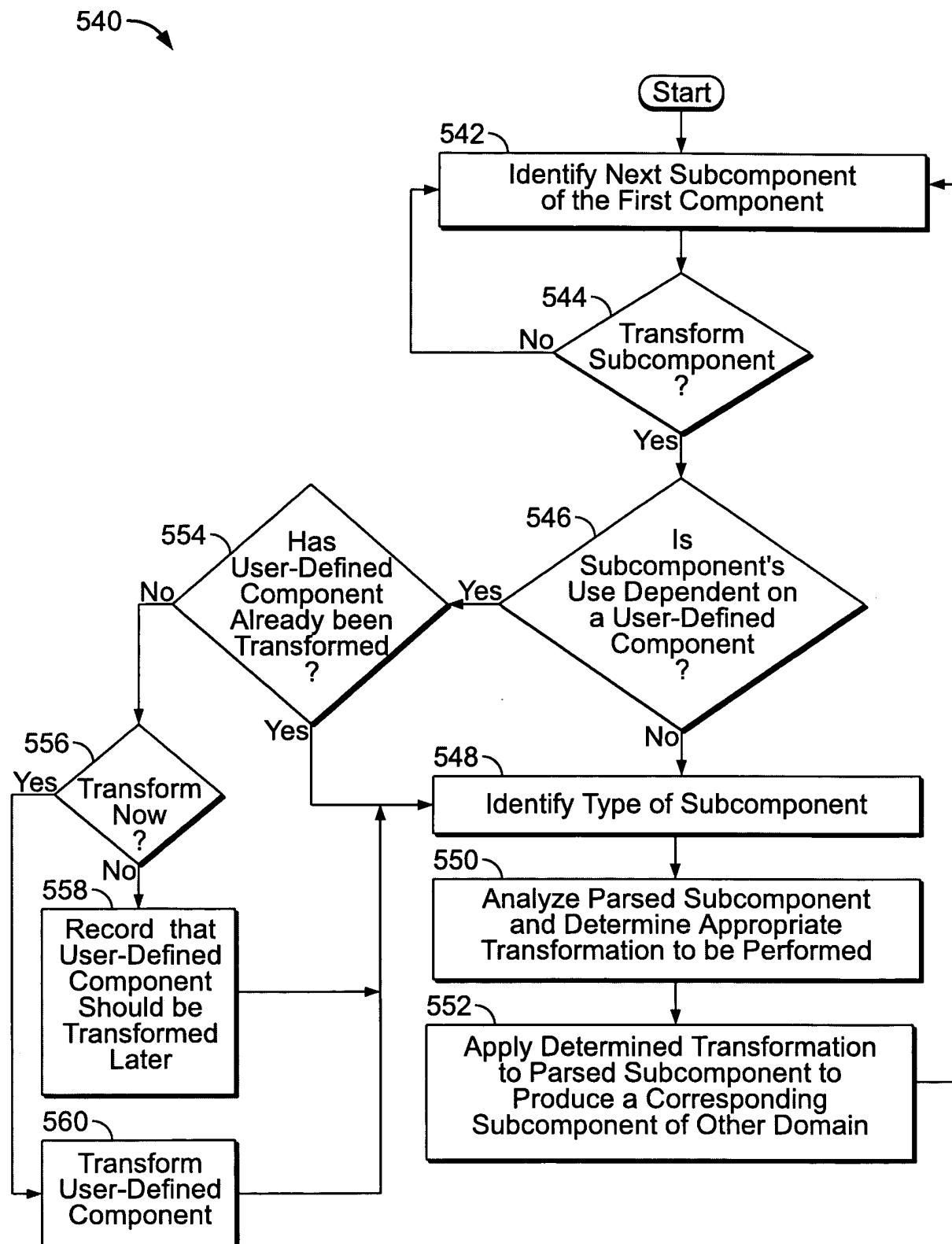


FIG. 21

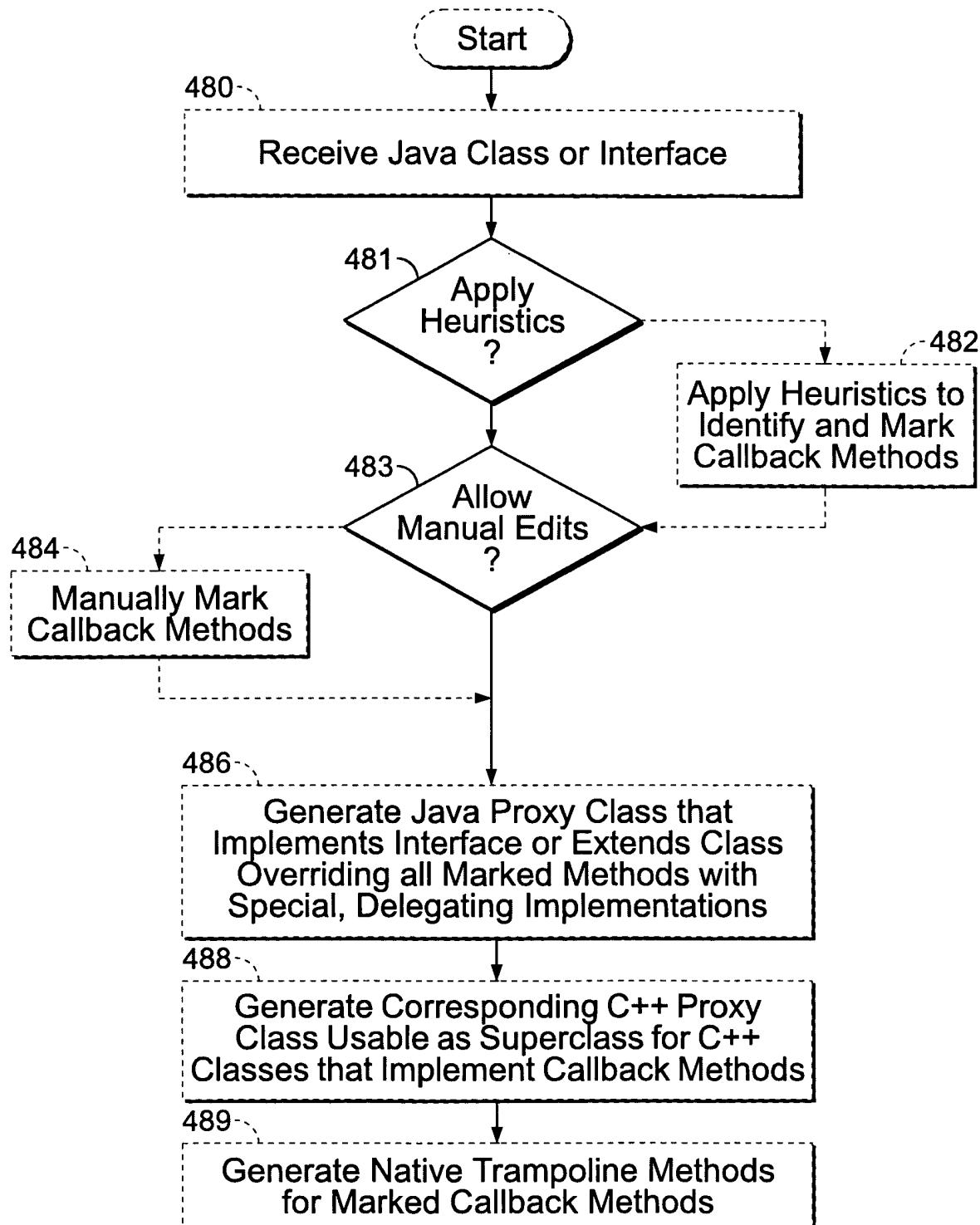


FIG. 22

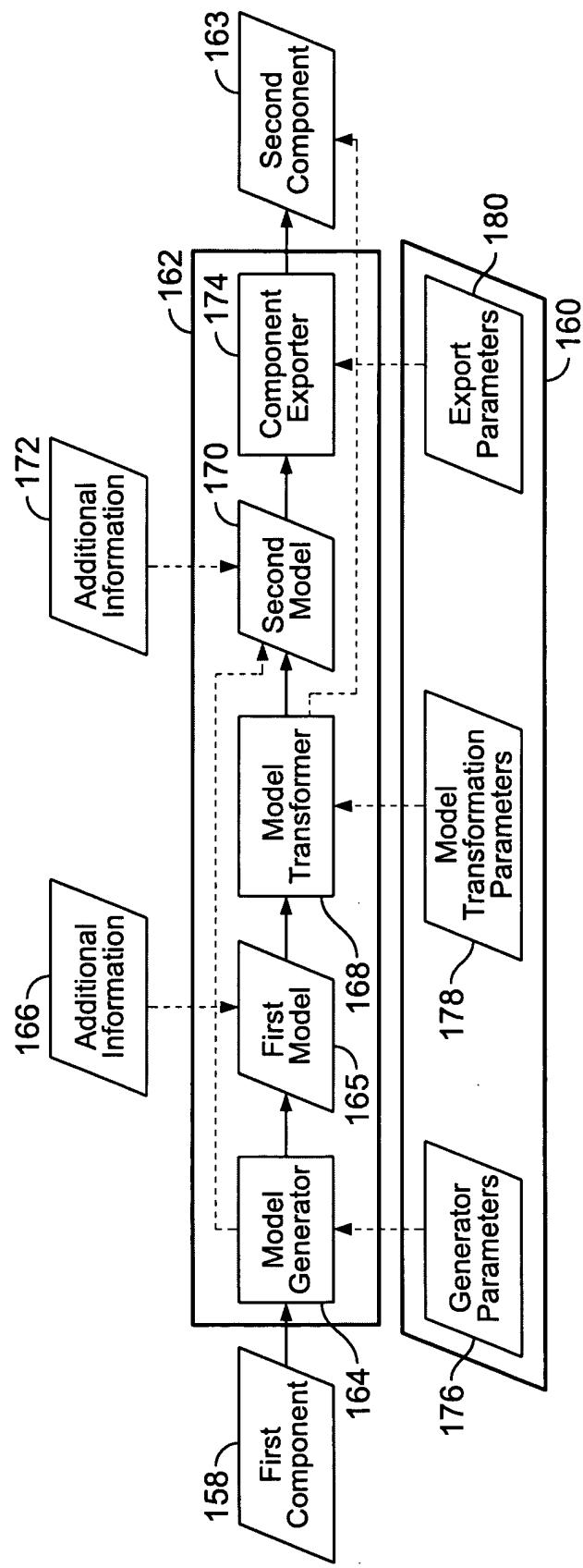


FIG. 23



300a →

```
public class Counter implements java.io.Serializable
{
    public static final int UP = 1;
    public static final int DOWN = 2;
    private int max;
    private int direction;
}

//creates a new UP-counter with the given maximum
public Counter( int _max )
{
    this( _max, UP );
}

//creates a new counter with given maximum and direction
public Counter( int _max, int _direction )
{
    max = _max;
    direction = _direction;
}

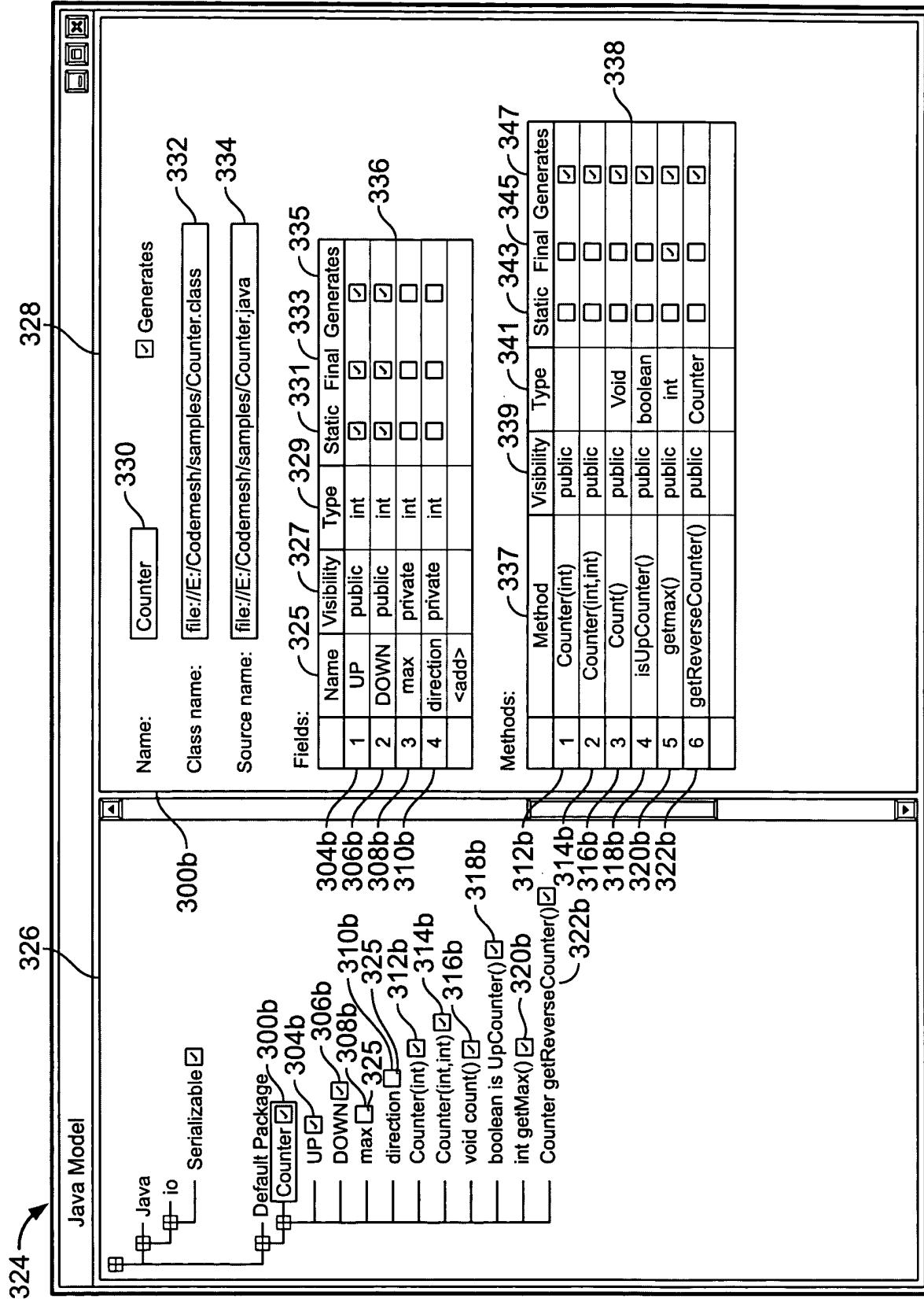
//counts in the direction specified and outputs the numbers
public void count()
{
    if( direction == UP )
        for( int l=0; l<max; l++ )
            System.out.println( " " + l );
    else if( direction == DOWN )
        for( int l=max-1; l>=0; l-- )
            System.out.println( " " + l );
}

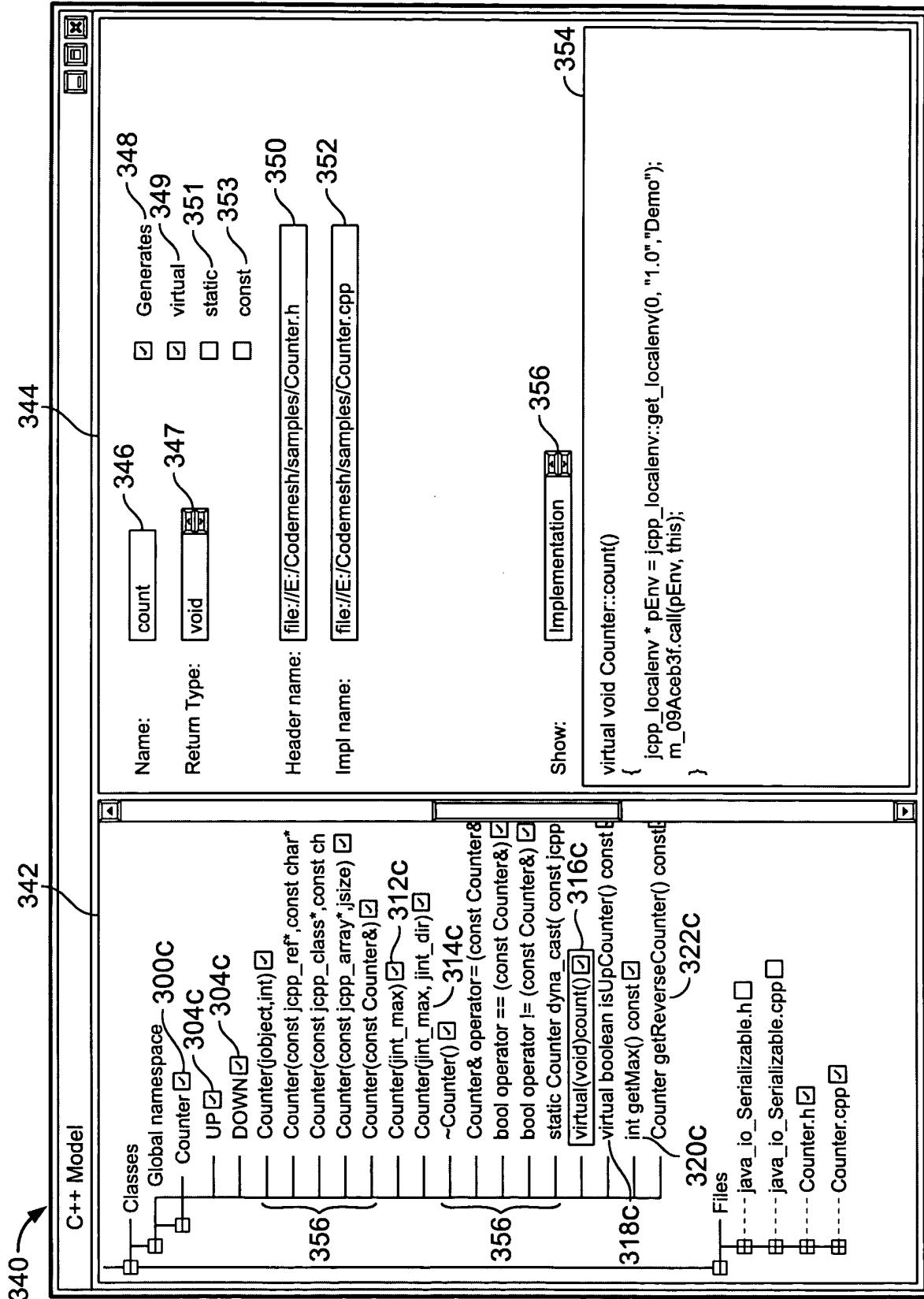
//returns true if this instance is an UP counter
public boolean isUpCounter()
{
    return ( direction == UP );
}

//returns the maximum of the counter
public final int getMax()
{
    return max;
}

//creates a counter with the same maximum as this counter, but reverse direction
public Counter getReverseCounter()
{
    return new Counter( max, direction == UP ? DOWN : UP );
}
```

FIG. 24





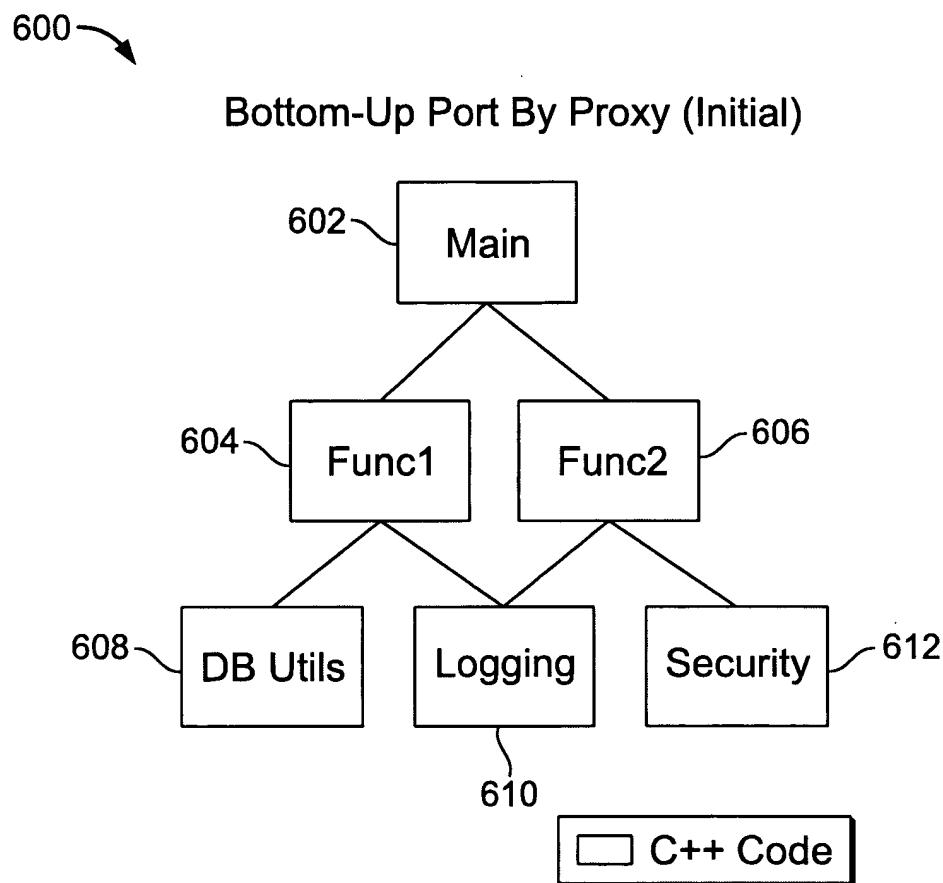


FIG. 27A



600 →
Bottom-Up Port By Proxy (1st Step)

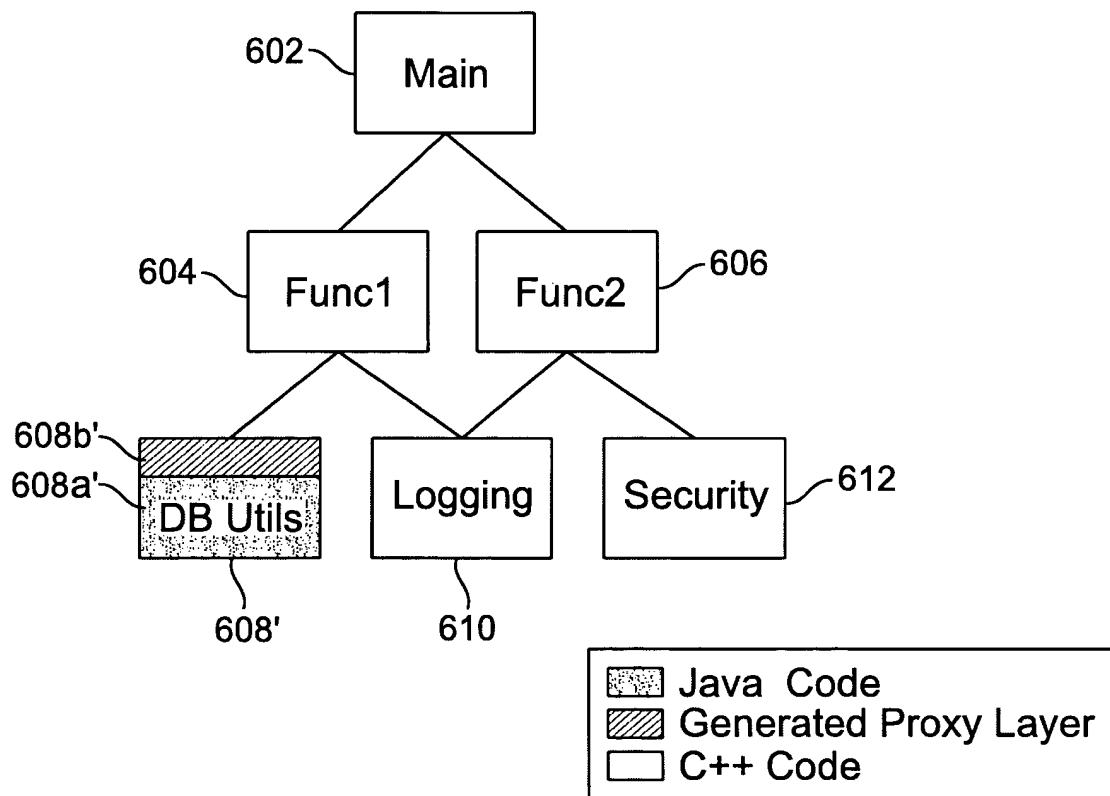


FIG. 27B



600 →
Bottom-Up Port By Proxy (2nd Step)

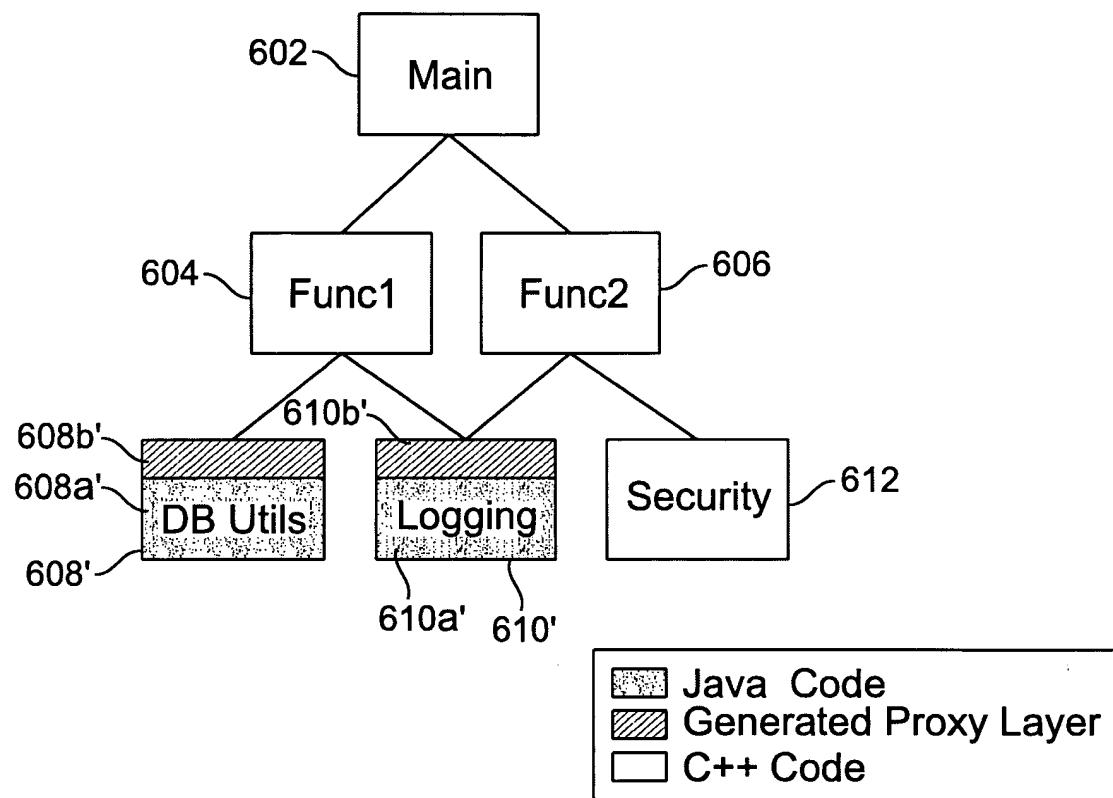


FIG. 27C



600

Bottom-Up Port By Proxy (3rd Step)

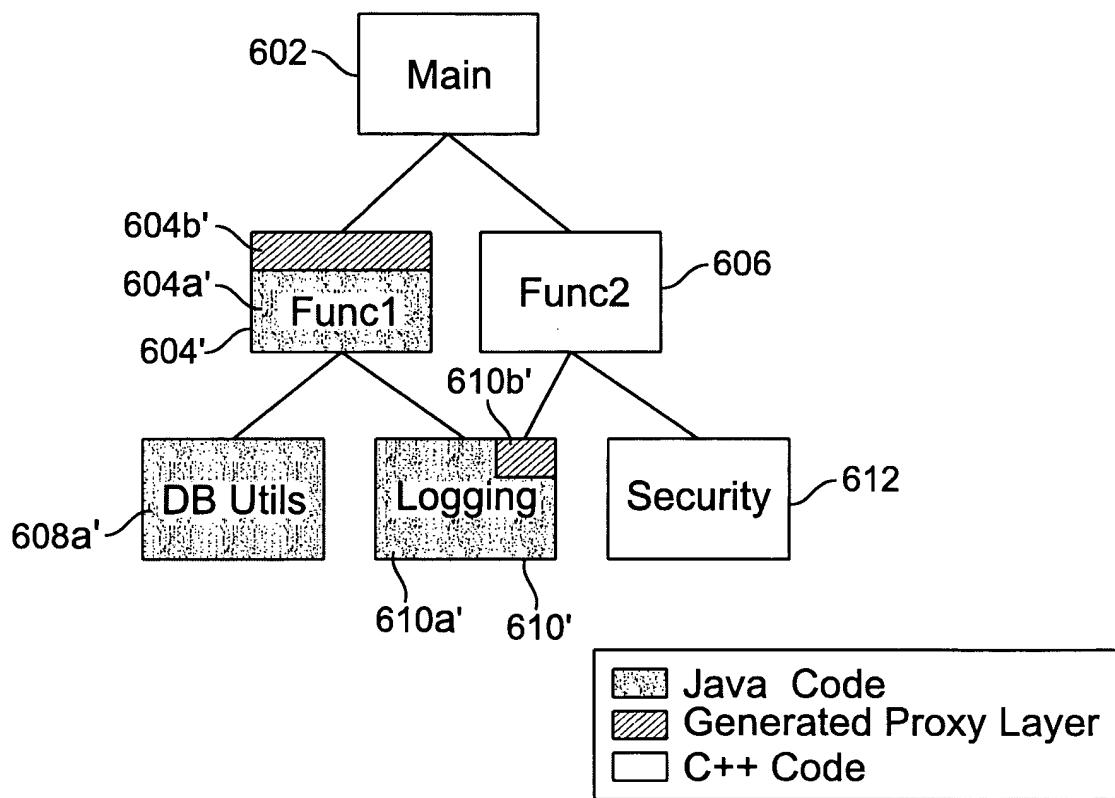
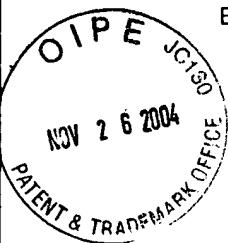


FIG. 27D



600 →
Bottom-Up Port By Proxy (4th Step)

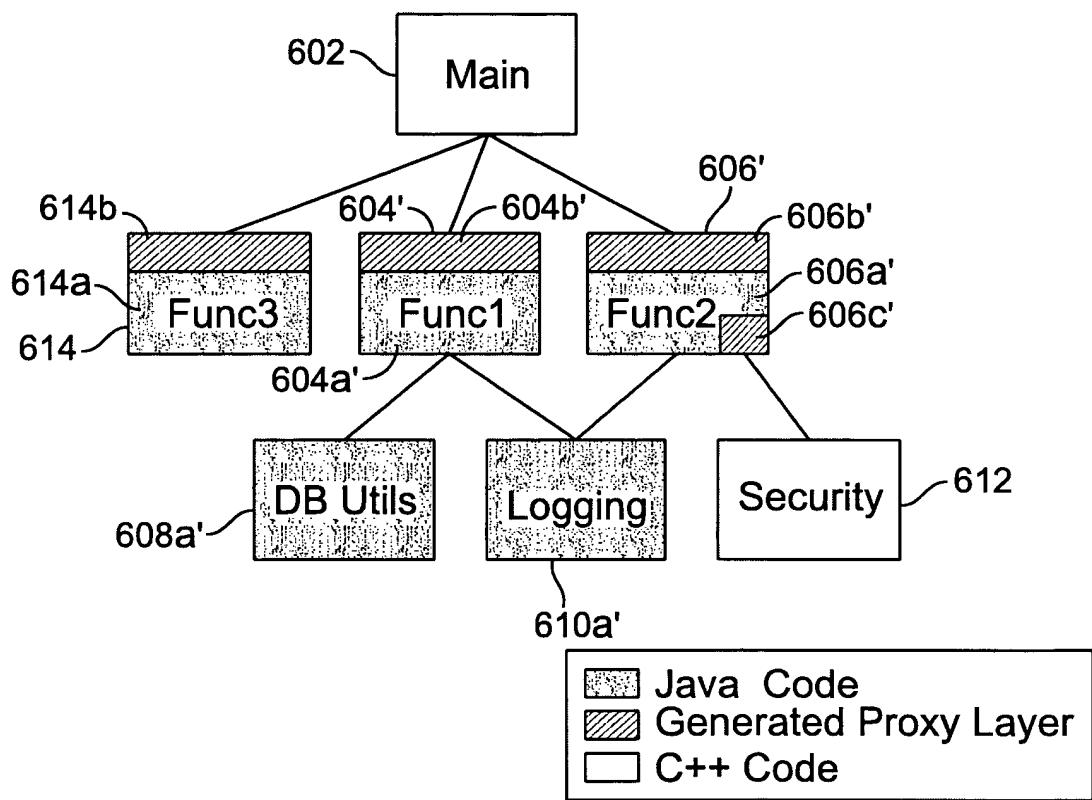
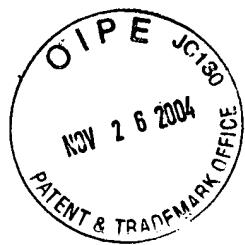


FIG. 27E



600

Bottom-Up Port By Proxy (5th Step)

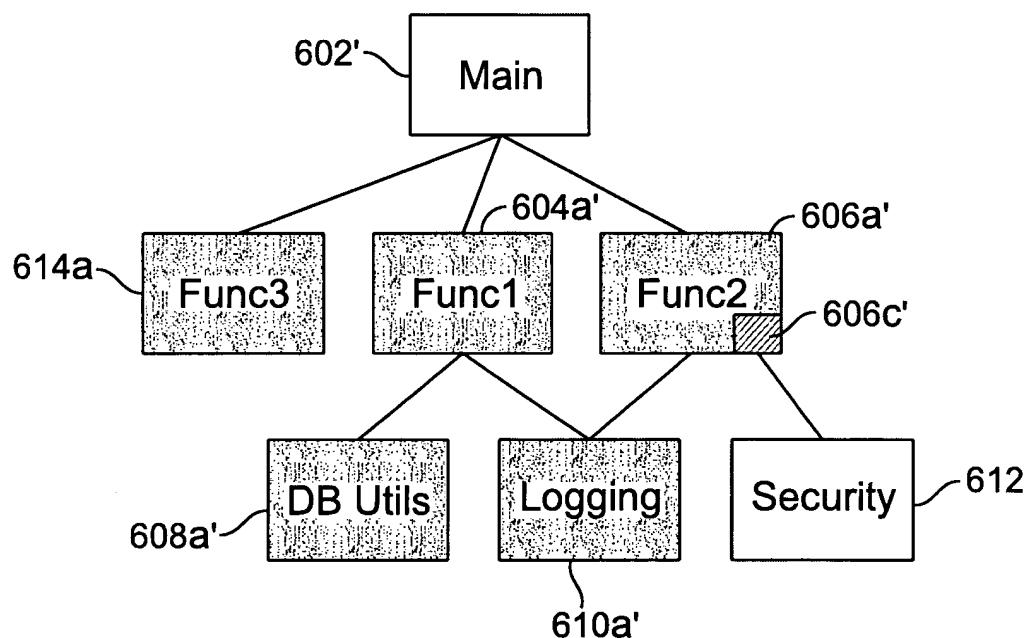


FIG. 27F